ONDERZOEKSRAPPORT NR. 8401

ON THE CALCULATION OF THE NUMBER OF FEASIBLE SEQUENCES IN ACTIVITY NETWORKS

J. VANTHIENEN

Onderzoeksrapport Nr. 8401

ON THE CALCULATION OF THE NUMBER OF FEASIBLE SEQUENCES IN ACTIVITY NETWORKS

J. VANTHIENEN

Wettelijk Depot : D/1984/2376/1

The Catholic University of Leuven, Belgium
Department of Applied Economic Sciences.

CONTENTS

0. INTRODUCTION

It is obvious that N elements (jobs, activities, ...) can be ordered in N! possible ways. However when precedence constraints are introduced, indicating that some elements should always precede some other elements, the calculation of the number of feasible sequences is no longer straightforward, as there is no single correspondence between the number of constraints and the resulting number of feasible sequences.

It is, however, often important to know the number of feasible sequences corresponding with a set of precedence constrained elements. Many enumerative methods, which examine each of the feasible sequences and select the optimal, would benefit from this knowledge as it offers the possibility to predict the total time needed to examine all sequences. In scheduling problems, e.g., the number of feasible alternatives might be so small that the best strategy is to enumerate them all. On the other hand, there may be so many alternatives, that looking for the optimal execution sequence by enumeration takes more time than the execution itself. The number of feasible sequences is also a measure of network complexity (ELMAGHRABY and HERROELEN [5], p. 230).

In this paper an algorithm is presented which enables the calculation of the number of feasible sequences for precedence constrained elements, as depicted in a precedence network. After a statement of the problem (section 1), a general solution procedure is described in section 2. The implementation of this algorithm is given in section 3 and some computational experiences are reported in section 4.

It is indicated in the last section that the implementation performs very well for problems which are not too complex. The algorithm was primarily designed for the calculation of the number of feasible condition orders in a decision table context in order to predict the time necessary to examine each of the sequences and select the optimal (with regard to

the number of table columns). As the number of elements (conditions) is less than 10 (see also VANTHIENEN [14]), the algorithm is fast enough to be used in an interactive microcomputer application.

The author wishes to acknowledge the helpful suggestions of Prof. Dr. W. HERROELEN and Prof. Dr. M. VERHELST, who also provided valuable comments on a draft of this paper.

1. PROBLEM STATEMENT

1.1. A simple example

Assuming N elements (jobs, tasks, activities, ...), there are N! possible sequences in which the N elements can be ordered (see e.g. BERMAN and FRYER [1]), as long as all elements are independent.

As an example, with N = 3 the 3! = 6 sequences are[1] :

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

Problems arise when precedence constraints have to be taken into account, i.e. when the elements are no longer independent. A precedence constraint indicates that a certain element x (predecessor) should always precede another element y (successor) (notation $x \lessdot y$).
In the presence of precedence constraints, the number of feasible sequences (i.e. sequences which satisfy all precedence relations) is smaller than the maximal value of N! and depends on the constraints.

As an example for N = 3 and $1 \lessdot 3$, $2 \lessdot 3$ only 2 of the 6 possible sequences are feasible (indicated with ▭) :

1 2 3	$1 \lessdot 3$, $2 \lessdot 3$ satisfied
1 3 2	$1 \lessdot 3$ satisfied, $2 \lessdot 3$ not satisfied
2 1 3	$1 \lessdot 3$, $2 \lessdot 3$ satisfied
2 3 1	$2 \lessdot 3$ satisfied, $1 \lessdot 3$ not satisfied
3 1 2	$1 \lessdot 3$, $2 \lessdot 3$ not satisfied
3 2 1	$1 \lessdot 3$, $2 \lessdot 3$ not satisfied

---

(1) N! (N factorial) = N. (N-1)! with 0! = 1.

When calculating the number of feasible sequences, a simple procedure would be to generate all possible sequences (see e.g. NIJENHUIS and WILF [9]), check the precedence constraints for each of them and count the feasible ones. This enumeration, however, would be so overwhelmingly time - and memory - consuming that it would hardly be applicable, except for very small problems.
Another approach would be to generate[1] and count all feasible sequences only, but this method is rather inefficient when only the number of feasible sequences is required and not their nature.

Moreover, as the number of feasible sequences largely depends on the nature of the precedence constraints, it would be more efficient to use this information when calculating the number of sequences. Note also, that (except in some special cases, see 2.1) no simple correspondence exists between the number of precedence constraints and the resulting number of feasible sequences.

1.2. Representation of precedence constraints

There are essentially two ways in which the precedence constraints can be represented : the precedence network and the precedence matrix. Although they both contain the same information, the former is able to present a visual interpretation of the problem while the latter offers considerable computational advantages.

1.2.1. Precedence network

When the elements are depicted as nodes and the precedence constraints as oriented arcs between two nodes, the resulting graph is called a precedence network, ((x)→(y) indicates $x \lessdot . y$: x should precede y) where each node can have multiple predecessors and multiple successors[2].

---

(1) A simple recursive algorithm for the generation of all feasible sequences is included in appendix B.

(2) Note that $x \lessdot . y$ does not mean that x and y can not be separated by any other elements. Consecutive elements which form a string should be combined in one node.

As an example, consider fig. 1 which illustrates a set of N = 4 elements with constraints $1 \lessdot . 3$, $2 \lessdot . 3$, $2 \lessdot . 4$:

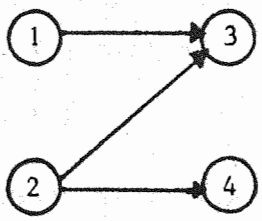


Fig. 1.

As precedence relations are transitive (i.e. : if $x \lessdot . y$ and $y \lessdot . z$ then $x \lessdot . z$) every node with both incoming and outgoing arcs would create redundant implied arcs connecting its predecessors and successors (fig. 2). For reasons of clarity, however, these redundant arcs are normally omitted and only direct (i.e. no implied) relations are shown (fig. 3).

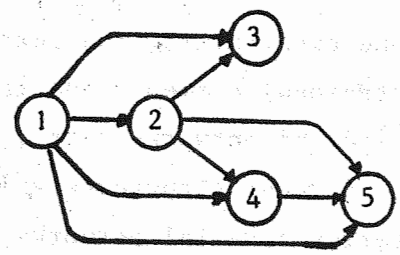


Fig. 2.

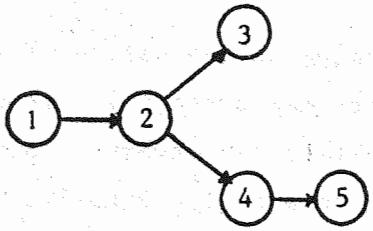


Fig. 3.

It has been assumed that at least one feasible sequence is present, i.e. there are no contradictory constraints.
If the network contains direct or implied loops, it is infeasible and the number of feasible sequences is zero.
E.g. - if $1 \lessdot . 2$ and $2 \lessdot . 1$ then $1 \lessdot . 1$: infeasible (fig. 4).
- if $1 \lessdot . 2$ and $2 \lessdot . 3$ and $3 \lessdot . 1$ then $1 \lessdot . 1$: infeasible (fig. 5).

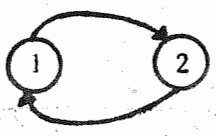


Fig. 4.

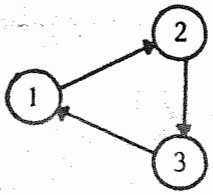


Fig. 5.

It has also been assumed that elements are given a number (label) according to the precedence constraints in order to avoid such confusing constructs as $y \lessdot . x$ where the number of element y is larger than the number of element x.

Though the proposed algorithm is able to deal with this unusual naming, networks as depicted in fig. 6 will not be considered.

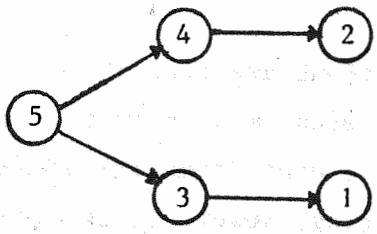


Fig. 6.

Depending on the nature of the precedence constraints, the corresponding network may contain independent (unconstrained) elements and independent (parallel) subnetworks. Some special kinds of networks exist, e.g. a chain or a tree. Finally, combinations of these structures are often encountered : parallel chains, forests, series parallel networks, ... (cfr. infra).

1.2.2. Precedence matrix

The precedence matrix $[p_{ij}]$ is an N x N square matrix of zeroes and ones, in which (cfr. fig. 7 and 8)

$$p_{ij} = \begin{cases} 1, \text{ if } i \lessdot . j \\ 0, \text{ otherwise} \end{cases}$$

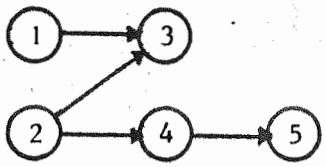


Fig. 7.

p_{ij}	1	2	3	4	5
1	0	0	1	0	0
2	0	0	1	1	1
3	0	0	0	0	0
4	0	0	0	0	1
5	0	0	0	0	0

Fig. 8.

In this matrix each 1 in row i indicates a successor of element i and each 1 in column j indicates a predecessor of element j.
It is common practice to include also implied constraints (e.g. $p_{2,5}$ in fig. 7) in the precedence matrix : if $p_{ij} = 1$ and $p_{jk} = 1$ then $p_{ik} = 1$[(1)]. This offers the advantage that all (immediate and implied) successors (predecessors) of an element are indicated in the corresponding row (column) of that element.

If contradictory constraints are present (direct or implied), any of the diagonal elements p_{ii} of the matrix will be 1.

E.g. - $1 < . 1$
- $1 < . 2$ and $2 < . 1$
- $1 < . 2$ and $2 < . 3$ and $3 < . 1$

$\Rightarrow p_{1,1} = 1$

As it is assumed that at least one feasible sequence is present (cfr. 1.2.1.) no diagonal element sould be 1.
Diagonal elements are, therefore, often indicated with - instead of 0, or are simply omitted.

As it is also assumed that elements never precede an element with a lower number (cfr. 1.2.1.), element labels are assigned such that the lower left part of the precedence matrix consists of zeroes and can therefore be deleted.

Incorporating these conventions (deletion of the lower left part and the diagonal of the matrix) leads to the following alternative notation (fig. 9) of the precedence matrix (fig. 8) :

---

(1) This can simply be done when entering the constraints : see section 3.1.

1	2	3	4	5	p_{ij}
	0	1	0	0	1
		1	1	1	2
			0	0	3
				1	4
					5

Fig. 9.

As there is a one-to-one correspondence between the precedence matrix and the precedence network, the same special configurations or combinations can be distinguished :

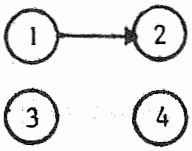
- independent elements : rows and columns of these elements contain all zeroes;
- independent (parallel) subnetworks : the precedence matrix can be divided in distinct submatrices which are unrelated.
- chain : all $\frac{N.(N-1)}{2}$ entries of the reduced precedence matrix are 1.
- tree : successively deleting elements with no predecessors creates parallel submatrices.
- ...

Given the complete correspondence between the precedence network and the matrix, attention will be focused on the visual aspects of the network and the computational advantages of the matrix will be postponed until the algorithm is presented in sufficient detail.

2. SOLUTION PROCEDURE

The number of feasible sequences can be calculated from the position matrix in the same way as the permanent function. Defining the position matrix $M = [m_{ij}]$ as an N x N square matrix (fig. 10) in which

$$m_{ij} = \begin{cases} 1, & \text{if element } i \text{ is permitted to occupy place } j \\ 0, & \text{otherwise} \end{cases}$$



$$M = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Fig. 10.

then the permanent function, perm (M), is calculated similar to the determinant except with no sign changes (see e.g. NIJENHUIS and WILF [9], p. 160). Consequently, for $N \geq 3$, the permanent function is computed recursively as the sum of the permanents of various submatrices, one for each element placed ahead in the sequence.

The permanent function is an upper bound to the number of feasible sequences (ELMAGHRABY and HERROELEN [6], p. 32). In order to get an exact count of the number of feasible sequences, the position matrix must be modified conditionally during the decomposition (as the place of some elements will be no longer restricted by the place of the elements which have already been put ahead in sequence).

As, however, the above calculation of the number of feasible sequences requires a recursive decomposition and a conditional updating of the position matrix, there is no need anymore to pass through the position matrix at all. The same operations can easily be performed on the precedence matrix itself. Elements permitted to occupy the first place are then simply elements with no predecessors (all zeroes in the corresponding column). Therefore, not the position matrix, but the precedence matrix (or equivalently the precedence network) will be used to calculate the number of feasible sequences.

For a few simple network structures, the number of feasible sequences is immediately known. More complex networks do not normally yield this direct solution, but often show a special structure which can be reduced rather easily to one or more simple structures. Only when the network (or reduced network) does not show a simple or special structure, it is decomposed (recursively) into various simultaneous subnetworks, placing ahead every element with no predecessors, one at the time, until simple or special structures are obtained.

2.1. Known solutions

It is obvious that when the network is unconstrained (fig. 11), the number of feasible sequences (S) equals the number of permutations (P)

$$S = N! \qquad (1)$$

Note also that for only one element the number of sequences equals 1, as no constraints are possible.

If the precedence network forms a single chain (fig. 12) there is only one feasible sequence : the order indicated by the chain :

$$S = 1 \qquad (2)$$

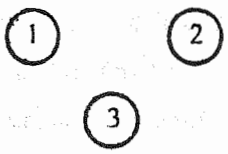


Fig. 11.

Fig. 12.

If there are only two elements, they are either unconstrained or they form a chain, so the number of feasible sequences equals 2! or 1[(1)].

Similarly, for three elements, only three new cases need to be distinguished (bearing in mind that implied arcs are not shown)(ELMAGHRABY and HERROELEN [6], p. 76) :

---

(1) As will be seen in 2.2.1. (special case 3), a network with only 1 constraint always shows N!/2 sequences.

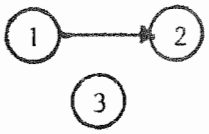


Fig. 13.

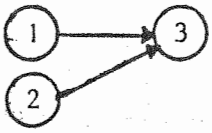


Fig. 14.

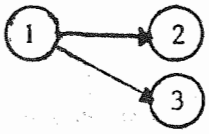


Fig. 15.

Direct computation shows that for three elements the number of arcs (including redundant arcs here) fully determines the number of feasible sequences

- 0 arcs : 6 sequences (unconstrained) : fig. 11.
- 1 arc : 3 sequences : fig. 13.
- 2 arcs : 2 sequences : fig. 14, 15.
- 3 arcs : 1 sequence (chain, 1 redundant arc) : fig. 12.

2.2. Special network structures

Some classes of networks show a structure which can easily be decomposed into parallel or serial subnetworks, for which the number of feasible sequences can be calculated independently.

2.2.1. Parallel networks

Let (N,R) be an N-node network (fig. 16), with precedence relation R, consisting of m parallel, unrelated subnetworks $(N_1,R_1),(N_2,R_2),\ldots(N_m,R_m)$.

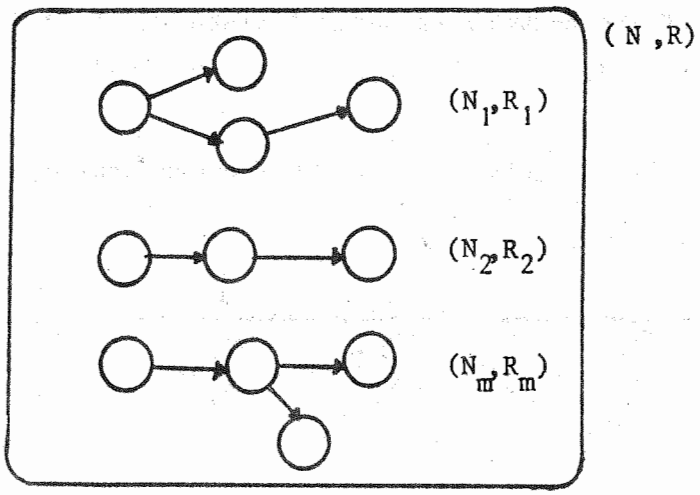


Fig. 16.

The total number of feasible sequences S_N for the network (N,R) is then given by (ELMAGHRABY and HERROELEN [5], p. 230).

$$S_N = \frac{N!}{\frac{N_1!}{S_{N_1}} \cdot \frac{N_2!}{S_{N_2}} \cdots \frac{N_m!}{S_{N_m}}} = N! / \prod_{i=1}^{m} (N_i/S_{N_i}) \qquad (3)$$

with S_{N_i} as short notation for $S_{(N_i,R_i)}$

As an example, consider the following network :

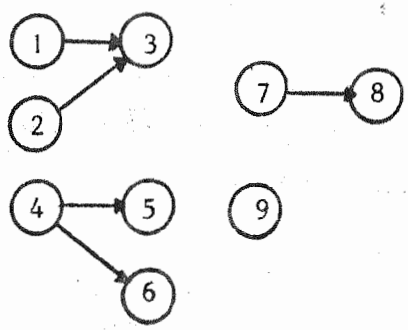


Fig. 17.

The network can be divided into 4 subnetworks, each of them having a known solution (see 2.1) :

$$S_9 = \frac{9!}{\frac{3!}{2} \cdot \frac{3!}{2} \cdot \frac{2!}{1} \cdot \frac{1!}{1}} = 20,160 \text{ feasible sequences}$$

Special case 1 (independent elements):

Note that if the network is unconstrained, all $N_i = 1$ and, therefore, all $S_{N_i} = 1$ (fig. 13). Hence Eq. 1 is obtained.

In general, if the network contains K unconstrained elements, then Eq. 3 becomes :

$$S_N = \frac{N!}{(\frac{N_1!}{S_{N_1}} \cdots \frac{N_K!}{S_{N_K}}) \cdot (\frac{N_{K+1}!}{S_{N_{K+1}}} \cdots \frac{N_m!}{S_{N_m}})}$$

$$= \frac{N!}{(N-K)!} \cdot \frac{(N-K)!}{(\frac{N_{K+1}!}{S_{N_{K+1}}} \cdots \frac{N_m!}{S_{n_m}})}$$

$$= \frac{N!}{(N-K)!} \cdot S_{N-K} \qquad (4)$$

This implies that the independent elements can be dropped and the total number of feasible sequences for the other elements has to be multiplied with N.(N-1) ... (N-K+1) or, shortly N!/(N-K)!

Note again, for N = K (unconstrained network), Eq. 4 reduces to Eq. 1 : $S_N = N!$

Special case 2 (parallel chain) :

A network (N,R) consisting of K parallel chains (N_1,R_1), (N_2,R_2) ... (N_K,R_K) has the property that each subnetwork has only one feasible schedule : $S_i = 1$, $i = N_1 \ldots N_K$ (Eq. 2).
It is then easy to see that (ELMAGHRABY and HERROELEN [5], p. 230)

$$S_N = \frac{N!}{N_1!.N_2!\ldots N_K!} = \left(\sum_{k=1}^{K} N_i\right)! \Big/ \prod_{k=1}^{K} N_K! \qquad (5)$$

Special case 3 (1 constraint) :

If a network (N,R) has only one constraint, it is immediately obvious that it consists of N-2 independent elements and a parallel chain of the two constrained elements. Therefore (Eq. 4).

$$S_N = \frac{N!}{(N-(N-2))!} \cdot 1 = \frac{N!}{2} \qquad (6)$$

As it is computationally rather difficult to detect multiple parallel networks, only two independent subnetworks are split at the time (fig. 18), each of them may then be subdivided again, until all parallel structures are detected (fig. 19).

So if a network (N,R) consists of two parallel networks (N_a,R_a) and (N_b,R_b) (where $N_b = N-N_a$), the numer of feasible sequences (Eq.3) simply reduces to :

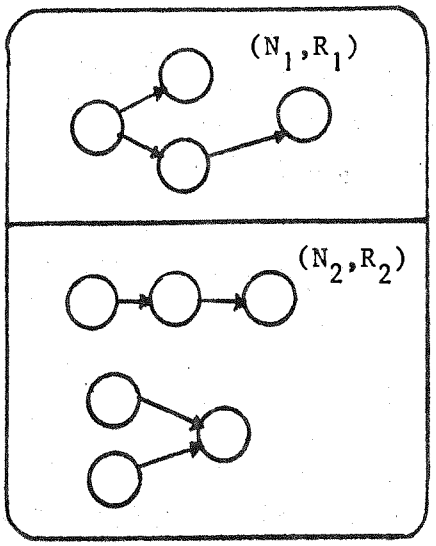


Fig. 18.

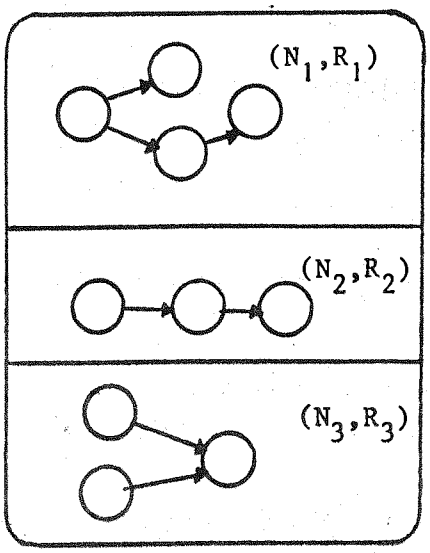


Fig. 19.

$$S_N = \frac{N!}{\frac{N_a!}{S_{N_a}} \cdot \frac{N_b!}{S_{N_b}}}$$

$$= \frac{N!}{N_a!.(N-N_a)!} \cdot S_{N_a} \cdot S_{N-N_a}$$

$$= \binom{N}{N_a} S_{N_a} \cdot S_{N-N_a} \qquad (7)$$

2.2.2. Serial networks

Let (N,R) be an N-node network (fig. 20), with precedence relations R, consisting of m subnetworks which form a chain (N_1,R_1), (N_2,R_2), ... (N_m,R_m), such that all elements of (N_i,R_i) precede all elements of (N_{i+1}, R_{i+1}).

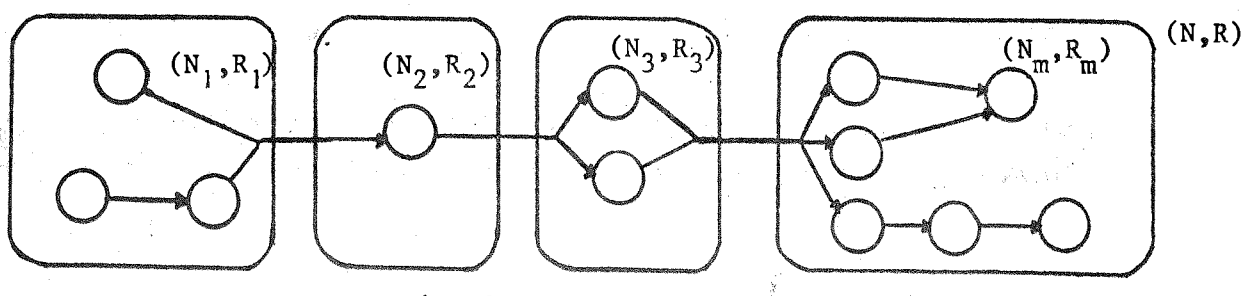


Fig. 20.

The total number of feasible sequences S_N for the network (N,R) is then given by (CONWAY et al. [4] , p. 242) :

$$S_N = S_{N_1} \cdot S_{N_2} \cdots S_{N_m} = \prod_{i=1}^{m} S_{N_i} \qquad (8)$$

Example :

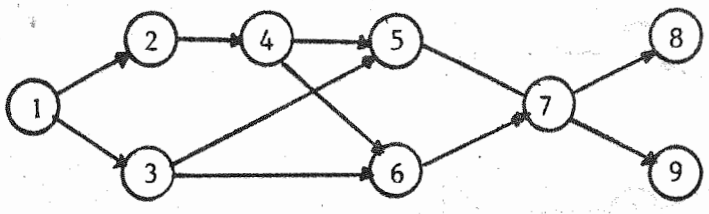


Fig. 21.

The network can be divided in 5 consecutive networks [1], [2 ,3,4] , [5,6], [7], [8,9], each of them having a known solution (see 2.1)(1) :

$$S_9 = 1!. \frac{3!}{2} . 2!.1!.2! = 12 \text{ feasible sequences}$$

Special case 1 (chain) :

In a chain every subnetwork has one single element and Eq. 2 is found :

$$S_N = 1.1... 1 = 1$$

Special case 2 (parent elements) :

If K elements bear identical constraints and have precedence over all other elements of the network (i.e. they all have N-K successors), then

$$S_N = K!.S_{N-K} \qquad (9)$$

If there is only 1 parent element (K=1), this element can be dropped or added arbitrarily as $S_N = S_{N-1}$.

---

(1) A (sub)network with given nodes and constraints will shortly be denoted as [enumeration of labels] . Eg. [1...N] = (N,R) and $S_{[1...N]} = S_N = S_{(N,R)}$.

Special case 2.1 (tree) :

A tree is a structure in which every element (except the first or parent) has exactly 1 (immediate) predecessor. (fig. 22).

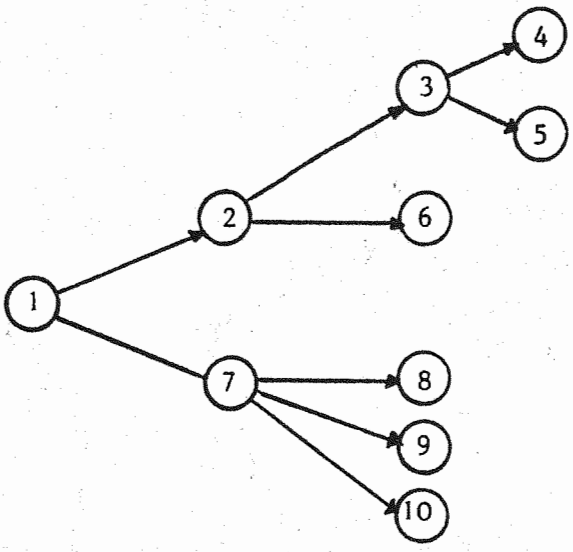


Fig. 22

Deleting the parent has no influence on the number of sequences (Eq. 9) and creates paralles subtrees (forest), to which the same procedure can be applied recursively. The total number of feasible sequences is therefore (Eq. 7) :

$$S_{[1\ldots10]} = \binom{9}{5} \cdot S_{[2\ldots6]} \cdot S_{[7\ldots10]}$$

$$\text{with } S_{[2\ldots6]} = \binom{4}{3} \cdot S_{[3\ldots5]} \cdot S_{[6]}$$

$$\text{with } S_{[3\ldots5]} = 2! = 2$$

$$S_{[6]} = 1$$

$$S_{[7\ldots10]} = 3! = 6.$$

which yields

$$S_{10} = \binom{9}{5} \cdot \binom{4}{3} . 2 . 1 . 6 = 6{,}048 \text{ feasible sequences.}$$

Special case 3 (terminal elements)

Similar to the parent elements case, K elements with identical constraints and all other N-K elements as predecessors, multiply the number of sequences with K! :

$$S_N = S_{N-K}.K! \qquad (10)$$

One terminal element (having N-1 predecessors) can be dropped or added arbitrarily.

Given the computational complexity of detecting serial networks, they are only split if K (one or more) identically constrained elements are successor or predecessors of all other elements (i.e. they all have the same N-K constraints[1]. (fig. 23.).

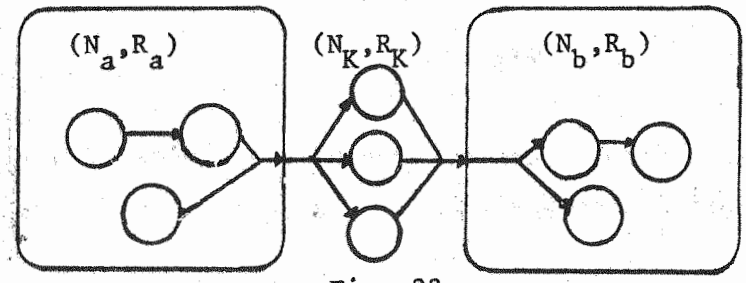


Fig. 23.

In that case

$$S_N = S_{N_a}.K!.S_{N_b} \qquad (11)$$

If the K elements are parent or terminal nodes, Eq. 11 becomes

$$S_N = K!.S_{N-K}$$

(1) The K elements are said to form a set (TONGE [12], p.27)

Note that for K=1, the corresponding element can simply be deleted from the (sub)network.

2.2.3. Example of a series parallel decomposition

The above calculations for serial, parallel or series-parallel structures are illustrated with an example (fig. 24) borrowed from ELMAGHRABY and HERROELEN [6], p. 84 (node labels have been rearranged for easy notation) :

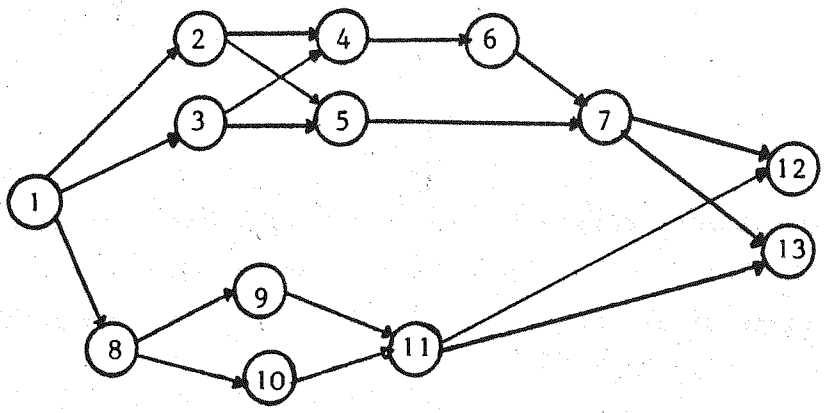


Fig. 25.

As node 1 is a parent (can be dropped), and node 12 and 13 are terminal elements (series decomposition),

$$S_{[1\ldots13]} = S_{[2\ldots11]} \cdot 2!$$

$S_{[2\ldots11]}$ now consists of two parallel subnetworks (fig. 26 and 27) :

$$S_{[2\ldots11]} = \binom{10}{6} S_{[2\ldots7]} \cdot S_{[8\ldots11]}$$

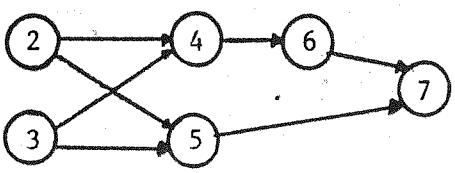


Fig. 26.

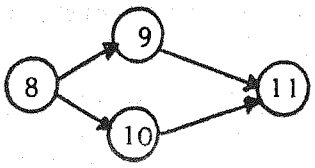


Fig. 27.

The first subnetwork (fig. 26) shows two parents and a terminal (series) element, leaving three elements with one constraint, which yields :

$$S_{[2...7]} = 2!.3.1 = 3!$$

The second subnetwork (fig. 27) is a serial network, hence the number of sequences equals :

$$S_{[8...11]} = 1.2!.1 = 2!$$

Totaling, the number of feasible sequences is given by :

$$S_{[1...13]} = \binom{10}{6}.3!.2!.2! = \frac{10!}{6!} = 5,040.$$

2.3. General solution procedure

As already illustrated in the previous example, the number of feasible sequences is calculated from the precedence network by detecting special solutions and structures and by recursively decomposing the network.

2.3.1. Solution elements for simple and series-parallel networks

The following solution elements are applied (see 2.1 and 2.2) :

- if the network has 0 or 1 constraints, the number of feasible sequences is N! or N!/2 resp. (Eq. 1 and 6)
- a chain has only 1 feasible sequence (Eq. 2)
- a network with 3 elements yields 6,3,2,1 sequences depending upon the number of constraints.
- K independent elements are dropped and the number of sequences should be multiplied with N!/(N-K)! (Eq. 4).
- 2 parallel networks (N_a,R_a) and (N_b,R_b) can be treated separately and the number of sequences equals the product of the number of sequences of the subnetworks multiplied with $\binom{N}{N_a}$ (Eq.7)

- K parent elements are dropped and the number of sequences should be multiplied with K! (Eq. 9)
- K terminal elements are dropped and the number of sequences should be multiplied with K! (Eq. 10)
- K elements which divide the network in two serial networks (N_a, R_a) and (N_b, R_b) are dropped and the number of sequences equals the product of the number of sequences of both subnetworks multiplied with K! (Eq. 11).

2.3.2. Normal decomposition

The above procedure is able to calculate the number of feasible sequences for small (≤ 2 constraints or ≤ 3 elements) or series-parallel networks.

The remaining problem is to deal with network structures as shown in fig. 28 : the so-called Z structures (TONGE [12] p. 28).

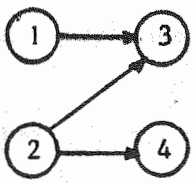


Fig. 28.

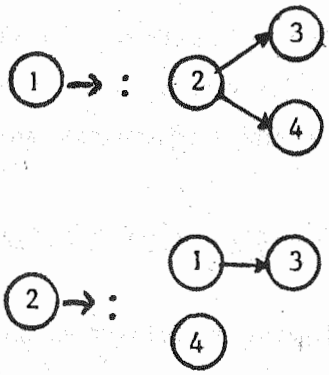


Fig. 29.

As this network does not show any of the special structures, the number of sequences can not be calculated immediately. It is necessary to choose each of the elements with no predecessors (one at the time), calculate the number of sequences for each subnetwork and finally add these results in order to obtain the number of feasible sequences for the complete network. For the network in fig. 28, the two subnetworks are given in fig. 29.

Performing these calculations for the networks in fig. 29 yields

$$S_{[1...4]} = S_{[2,3,4]} + S_{[1,3,4]} = 2 + \frac{3!}{2} = 5 \text{ sequences}$$

Note that this decomposition (fig. 29) fully corresponds to the complete enumeration of possible element sequences (cfr. the tree diagram in fig.30):

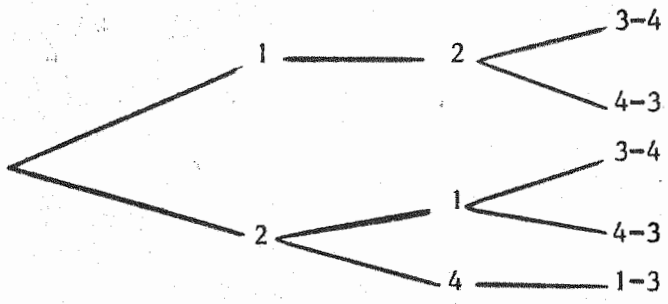


Fig. 30.

It is obvious that this so-called normal decomposition is a time-consuming solution, hence it should be avoided as much as possible, by detecting the special network structures.

If during the normal decomposition of elements with no predecessors, K of these elements are found to have the same successor, only 1 of the K decompositions is necessary, and the number of sequences can simply be multiplied by K, as all K elements are perfectly substitutable.

$$S_N = S_{N-1} + \underbrace{S'_{N-1} + S'_{N-1} \cdots S'_{N-1}}_{K}$$

$$= S_{N-1} + K.S'_{N-1} \qquad (12)$$

E.g.

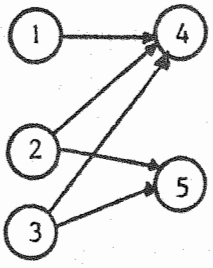


Fig. 31.

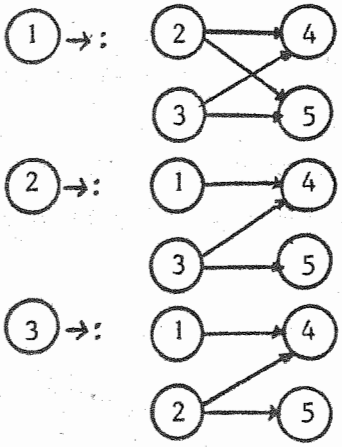


Fig. 32.

$$S_{[1\ldots5]} = S_{[2\ldots5]} + S_{[1,3,4,5]} + S_{[1,2,4,5]}$$

$$= S_{[2\ldots5]} + 2.S_{[1,3,4,5]}$$

$$= 4 + 2.5$$

$$= 14 \text{ sequences}$$

The general procedure for calculating the number of feasible sequences is therefore : look if the network has some simple and known solution. If not, try to find special structures. If none is found, decompose the network for every element with no predecessors and restart the procedure for each subnetwork[1].

This procedure is illustrated in the next section.

---

(1) Given the complex nature of normal decomposition, the number of special structures and the number of decompositions encountered when calculating the number of feasible sequences is a measure of network complexity.
It is obvious that networks consisting of e.g. trees and parallel chains are less complex than networks which have to be decomposed because no series parallel structure is found.

2.3.3. Illustrative example

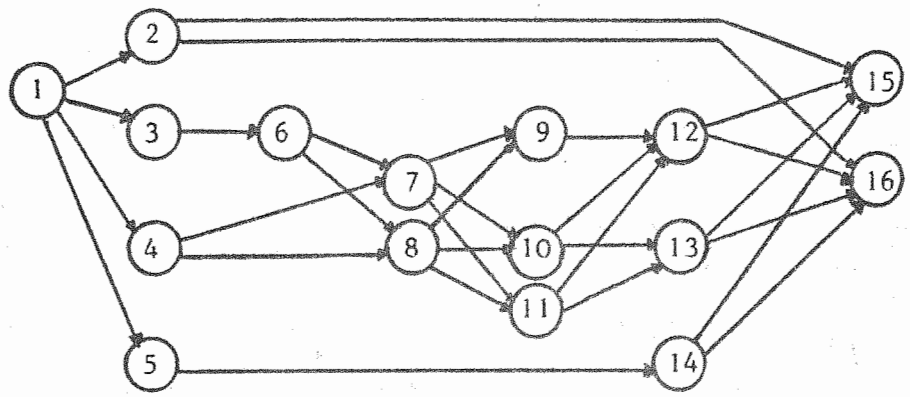


Fig. 33.

$S_{[1...16]} = S_{[2...14]} \cdot 2$ (1 parent (1), 2 terminal elements (15) (16))

$S_{[2...14]} = \frac{13!}{12!} \cdot S_{[3...14]}$ (independent element (2))

$S_{[3...14]} = \binom{12}{10} \cdot S_{[5,14]} \cdot S_{[3,4,6...13]}$ (parallel decomposition (5))

$S_{[5,14]} = 1$ (parent (5), independent (14))

$S_{[3,4,6...13]} = S_{[3,4,6]} \cdot 2! \cdot S_{[9...13]}$ (series decomposition (7) (8))

$S_{[3,4,6]} = \frac{3!}{2}$ (simple network (3) (4) (6))

$S_{[9...13]} = S_{[10...13]} + 2 \cdot S_{[9,11...13]}$ (normal decomposition (9) (10) (11))

$S_{[10...13]} = 2! \cdot 2!$ (2 parents (10) (11), 2 independent elements (12) (13))

$S_{[9,11...13]} = S_{[11...13]} + S_{[9,12,13]}$ (normal decomposition (9) (11))

$S_{[11...13]} = 2$ (simple network (11) (12) (13))

$S_{[9,12,13]} = 3$ (simple network (9) (12) (13))

Yielding :

$$S_{[9,11\ldots13]} = 2 + 3 = 5$$

$$S_{[9\ldots13]} = 4 + 2,5 = 14$$

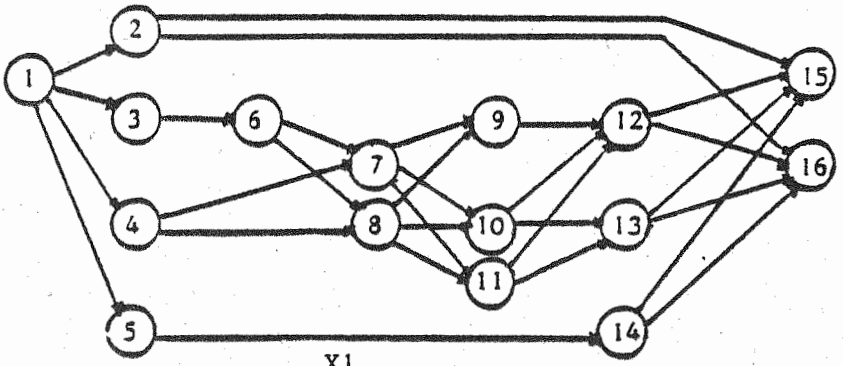
$$S_{[3,4,6\ldots13]} = 6,14 = 84$$

$$S_{[3\ldots14]} = \binom{12}{10}.84 = 5,544$$

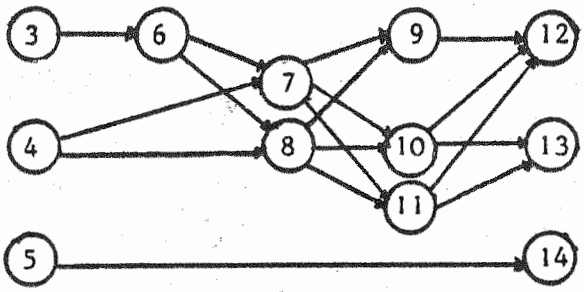
$$S_{[2\ldots14]} = 13.\ 5,544 = 72,072$$

$$S_{[1\ldots16]} = 72,072.2 = 144,144 \text{ feasible sequences.}$$

The solution procedure is shown in fig. 34.



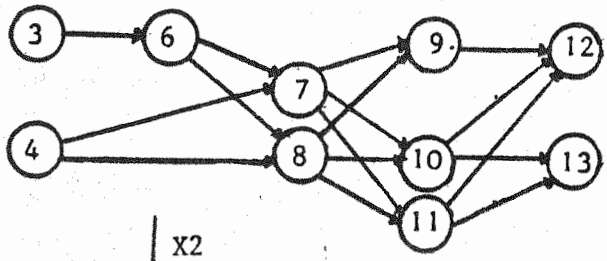
X1
X2
X13



X66

X

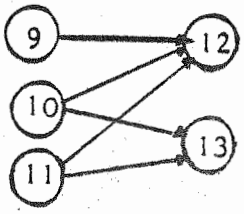
1



X2

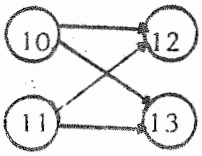
X

3

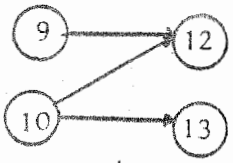


\+

X2



:



:

3. DESCRIPTION OF THE ALGORITHM

The solution procedure of 2.3 has been implemented in a recursive[1] algorithm, which is shown in 3.1. Computational results with this algorithm will be given in section 4.

3.1. Pseudo code

Not all computational details of the algorithm are presented here, but only the high level structure in pidgin algol notation. Though the complete algorithm is written in PASCAL, translation from this pseudo code to PASCAL is very straightforward.

The algorithm proceeds as follows :
somewhere in the main program the precedence network (matrix) is read and implied constraints are added[2].
If the problem is feasible, the number of feasible sequences for the given number of elements is calculated in function NUMBER.
If necessary the network is decomposed and recursive entries to NUMBER are made.
Fig. 35 shows the code for this algorithm.

3.2. Overview of solution elements

The number of feasible sequences is calculated from the precedence network by applying the solution elements of section 2.3.

Many of these elements ask for a recursive decomposition of the original network, but in order to avoid the overhead of an unnecessary recursive call, it is first attempted to drop as many elements as possible from the considered network, such that recursion is only applied if at least two subnetworks are required.

---

(1) A recursive algorithm is an algorithm which relates the problem to a smaller problem and calls itself until the problem is simple enough to be solved immediately.
(2) Their only use is to facilitate the detection of parallel subsets (see 3.2).

```
PROGRAM sequences;
{feasible sequences for N elements (J. Vanthienen, 1983)}

  FUNCTION number(precedence constraints: no. of elements):real; {recursive}
  {calculate number of feasible (sub)sequences with this no. of elements}
  BEGIN {number}
    no. of sequences := 0;
    free levels := 1;
    REPEAT {trying to drop some elements}
      CASE {remaining} no. of constraints OF
        0 : no. of sequences := no. of elements !;
        1 : no. of sequences := no. of elements ! / 2;
        (no. of elements * (no. of elements - 1)) / 2 : no. of sequences := 1;
        OTHERWISE
          IF no. of elements = 3
            THEN no. of sequences := 2
            ELSE FOR all elements DO
                   BEGIN IF unconstrained element
                           THEN BEGIN free levels := free levels * no. of elements;
                                      drop element
                                END;
                         IF some elements are followed only by all others
                           THEN BEGIN free levels := free levels * no. of these elements !;
                                      drop these elements
                                END;
                         IF some elements are preceded only by all others
                           THEN BEGIN free levels := free levels * no. of these elements !;
                                      drop these elements
                                END
                   END
      END
    UNTIL no more elements have been dropped;
    IF no. of elements with no predecessors > 1
      THEN IF some elements separate the complete network
             THEN {divide serial subsets}
                  no. of sequences := no. of these elements !
                                      * number(constraints.no. of predecessors of these elements)
                                      * number(constraints.no. of successors of these elements)
             ELSE IF any of the elements with no predecessors is independent from the others
                    THEN {divide parallel subsets}
                         no. of sequences := combinations(no. of elements.independant element with successor
                                             * number(constraints.independant element with successors)
                                             * number(constraints.other elements)
                    ELSE IF no. of elements = 4 {Z structure}
                           THEN no. of sequences := 5
                           ELSE {decompose the network for all elements without predecessors}
                                FOR every untreated element with no predecessors DO
                                  no. of sequences := no. of sequences + no. of equal elements
                                                      * number(constraints.no. of elements - this element);
    number := no. of sequences * free levels
  END; {number}

BEGIN {sequence}
  READ (no. of elements);
  WHILE constraints are entered DO
    BEGIN READ (element x should precede element y);
          BEGIN {add implied constraints}
                x and predecessors of x should precede y and successors of y
          END
    END;
  IF any element should precede itself (originally or implied)
    THEN no. of sequences := 0 {automatically infeasible}
    ELSE no. of sequences := number(precedence constraints.no. of elements);
  WRITE ('number of sequences =', no. of sequences)
END. {sequence}
```

Fig. 35.

The algorithm, therefore, consists of two parts :

- Drop elements if possible
- Decompose the network in (at least two) subnetworks if necessary.

If less than 2 constraints or less than 4 elements remain or if the network is decomposed the algorithm has reached an end.

Part I : dropping elements

- Networks with 0 or 1 constraints (including networks with 1 or 2 elements), offer an immediate solution (Eq. 1,6)
- If the precedence matrix contains $\frac{N.(N-1)}{2}$ ones, all elements form a chain (Eq. 2).
- Networks with 3 elements, not having 0, 1 or 3 constraints, must have 2 constraints and, hence, 2 sequences (see 2.1)
- Unconstrained elements and terminal elements are dropped after the appropriate multiplication(1) of the number of sequences (Eq. 4, 10).
- Parent elements (being predecessors of all other elements) would call for a recursive decomposition, but as all decompositions are similar, only one of them would be executed (Eq. 9), yielding a one parent case. If there is only one parent, the corresponding element can be dropped immediately.

Note that dropping an element (e.g. a parent) could make other elements ready for deletion (e.g. the new parent). As those elements might have been examined already(2), it is necessary to repeat the examination until no more elements have been dropped.

Part II : Recursive decomposition

If no single solution was found and at least two elements without predecessors can be chosen, the network has to be decomposed.

---

(1) Note that his multiplication can only be performed after the number of sequences is calculated.

(2) Note that elements need not be numbered in an order which satisfies the constraints.

- first a check is made for serial networks: equal elements with all other elements as predecessors or successors create two serial subnetworks, which can be treated separately (Eq. 11).
- parallel subnetworks are harder to detect[1] : an element (with no predecessors) is independent from all other elements (with no predecessors) if all predecessors of the successors of that element (immediate or implied) are also successors of that element. Note that only one element with no predecessors can be separated this way. The number of schedules is then calculated according to Eq. 7.
- If no series or parallel decomposition was possible, normal decompositions are executed, one for each class of equal elements (Eq. 12), unless the network contains 4 elements, in which case 5 sequences are feasible (fig. 29)[2].

3.3. Computational considerations

Implementation of this algorithm is done by using the precedence matrix. But as dropping an element implies a time-consuming row and column deletion from the matrix, elements are not deleted but marked (with a negative number) in a vector of column totals which also keeps track of the number of ones in every column.

This procedure would leave the precedence matrix unchanged. It is, however, clear that it would not be very efficient to work with a precedence matrix which contains an overhead of 90 % deleted rows and columns. Therefore the matrix is periodically cleaned up, by deleting all marked rows and columns[3].

---

(1) In order to detect parallel subsets, one could easily check the intersection of successor sets. The implementation of the PASCAL set construct, however, is too limited to handle a large number of elements.
(2) Explicit enumeration shows that the remaining cases for the 4 element problem can only be Z structures.
(3) Some educated guesses showed that it is worthwile to do this when a matrix with more than 10 elements enters the function with at least 20 % of its elements marked.

Most of the execution time is spent looking for serial and parallel subsets. Improvement of these detection techniques would constitute a major acceleration.

Finally, immediate calculation of more z structures, without the need for recursive decomposition, would favor execution efficiency (e.g. a 5 element network with a Σ structure is known to yield 16 feasible sequences, fig. 36).

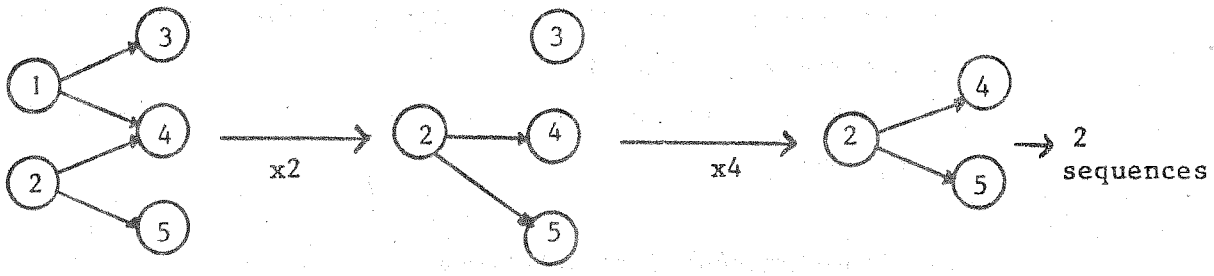


Fig. 36.

4. COMPUTATIONAL RESULTS

The above algorithm for the calculation of the number of feasible sequences in activity networks has been applied to a few networks found in the literature.

In table I the list of networks is presented (the networks are depicted in appendix A in this order, fig. A.1 - A.14 :

The algorithm was written in PASCAL for the non-optimizing PASCAL 8000, Version 2.0, compiler, and run (without execution tests) on an IBM 3033 computer. Results and CPU times (measured immediately before and after the function call) are displayed in table II[1].

(1) The same algorithm, written in PASCAL/M, was also run on a 64K CD110 microcomputer under CP/M 2.2, via PASCAL/M (which generates Pcode). Results indicated that execution time was approximately 1000 times larger. E.g. the 21 element problem took 348 seconds instead of 318 milliseconds (Execution tests, however, were included here).

TABLE I : Problems

Network	No. of elements	Total no. of constraints	Source
1	4	1	Elmaghraby and Herroelen [6], p. 63
2	5	5	Reeve [11], p. 76
3	7	19	Reeve [11], P. 74
4	7	16	Reeve [11], p. 84
5	8	14	Elmaghraby and Herroelen [6], p. 63
6	9	30	Moodie [8], p. 9
7	11	42	Moodie [8], p. 36
8	11	32	Reeve [11], p. 5
9	13	49	Elmaghraby and Herroelen [6], p. 64
10	19	82	Moodie [8], p. 153
11	21	149	Reeve [11], p. 70
12	45	429	Moodie [8], p. 159
13	48	735	Moodie [8], p. 163
14	70	1435	Moodie [8], p. 173

TABLE II : Results

Network	No. of elements	Total no. of constraints	No of feasible sequences	CPU time
1	4	1	12	< 0.001 secs.
2	5	5	12	< 0.001 secs.
3	7	19	4	< 0.001 secs.
4	7	16	10	< 0.001 secs.
5	8	14	140	< 0.001 secs.
6	9	30	24	< 0.001 secs.
7	11	42	97	0.008 secs.
8	11	32	956	0.001 secs.
9	13	49	5040	0.001 secs.
10	19	82	1937295360	0.003 secs.
11	21	149	1449624	0.318 secs.
12	45	429	(*)	(*)
13	48	735	65135164675891628.10^{8}	4.030 secs.
14	70	1435	(*)	(*)

(*) These networks could not be solved in 1 hour of CPU-time and were, therefore, cancelled before completion.

From these execution times, it is obvious that the algorithm performs very well for networks which are not too complex. Large networks, containing few series parallel or other special structures, and, hence, requiring many normal decompositions are harder to solve. Further improvement of the algorithm, however, as indicated in 3.3, would favor execution efficiency for these problems.

APPENDIX A : Problem examples

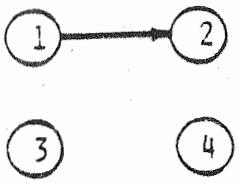


Fig. A.1.: 4 Element Problem

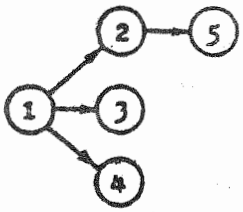


Fig. A.2.: **5 Element Problem**

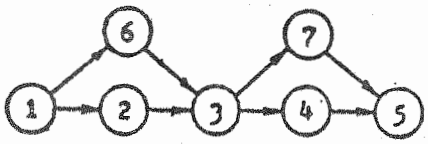


Fig. A.3. : **7 Element Problem**

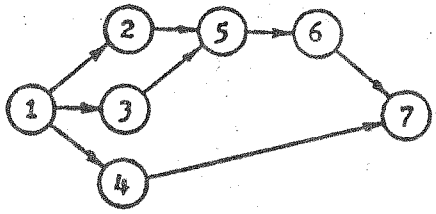


Fig. A.4.: 7 Element Problem

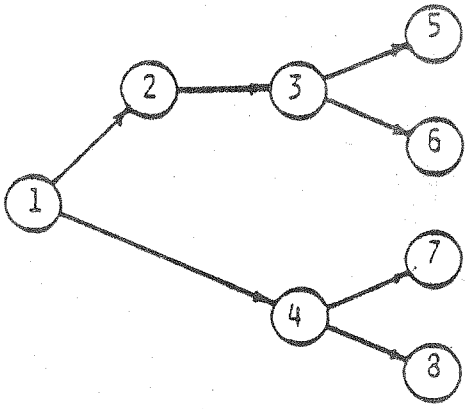


Fig. A.5.: Initially Rooted Tree

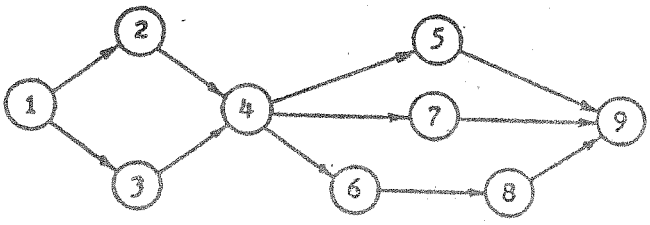


Fig. A.6. : 9 Element Problem

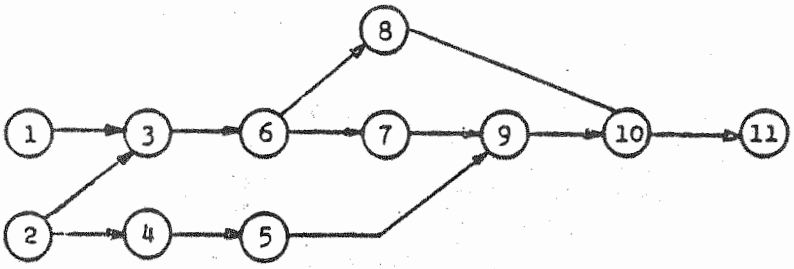


Fig. A.7. : 11 Element Problem

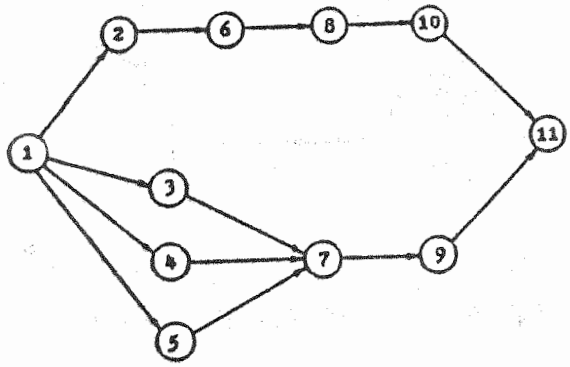


Fig. A.8. : 11 Element Problem

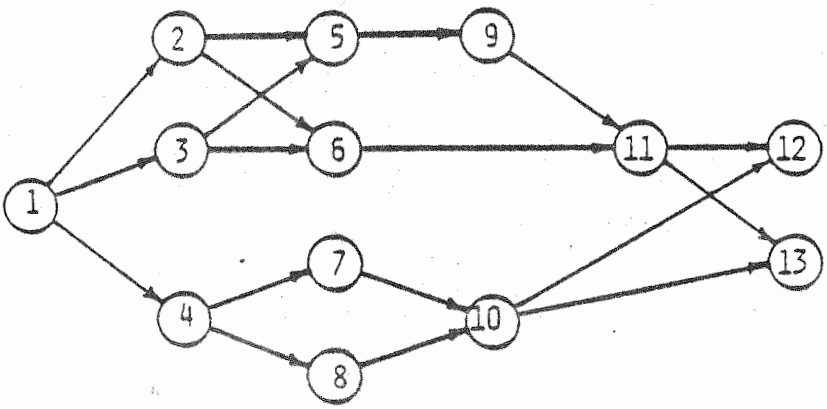


Fig. A.9. : Series Parallel Digraph

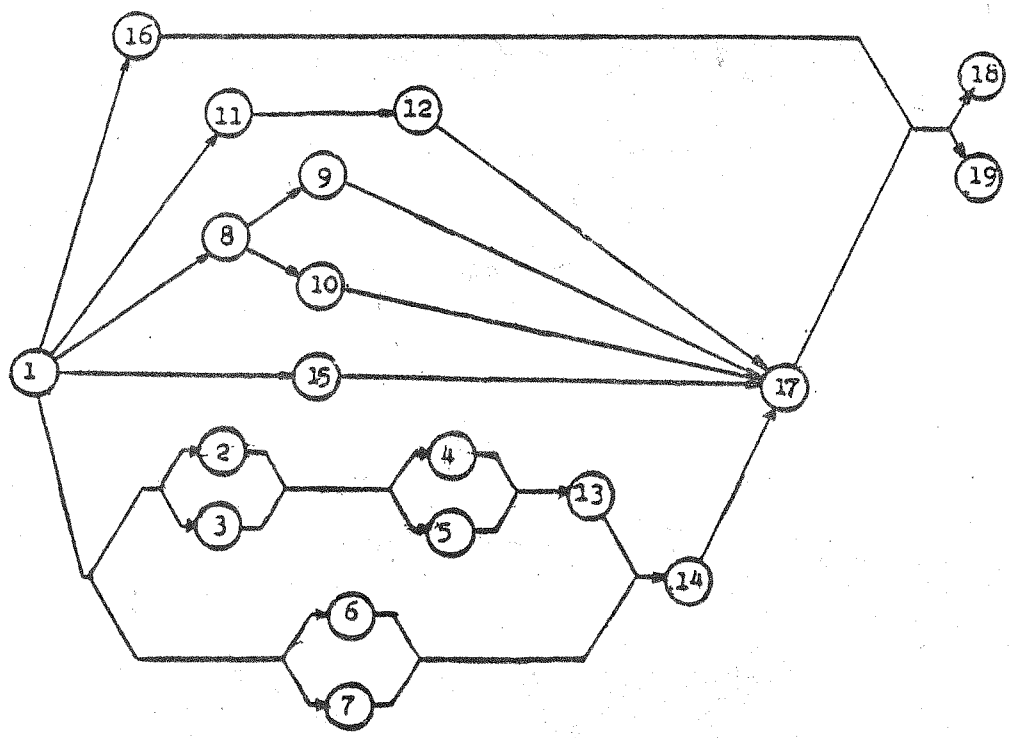


Fig. A.10. : 19 Element Problem

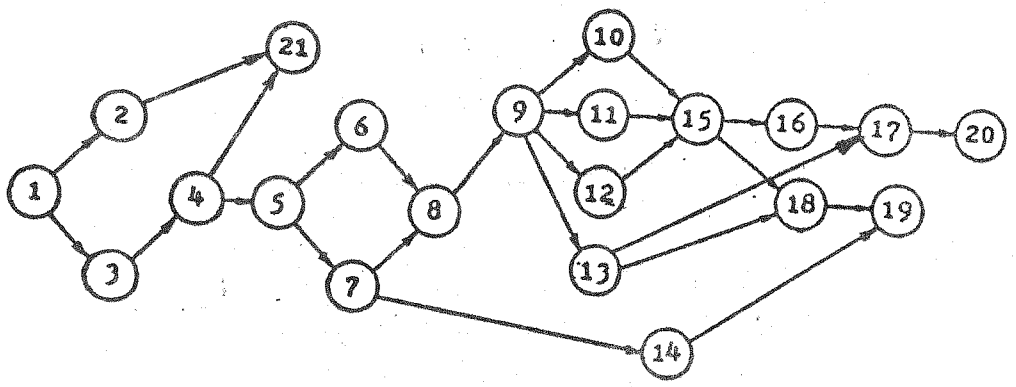


Fig. A.11. : 21 Element Problem

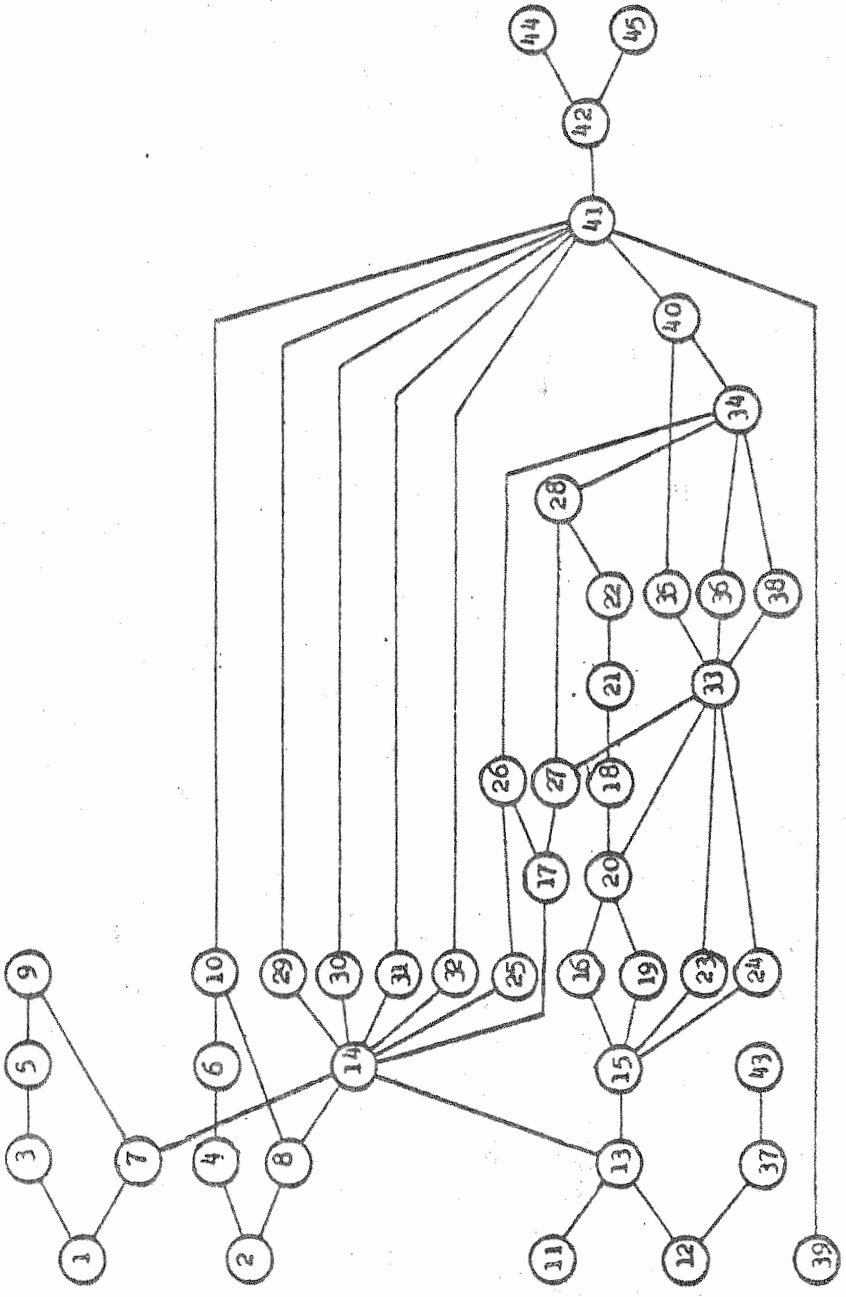


Fig. A.12. : 45 Element Problem

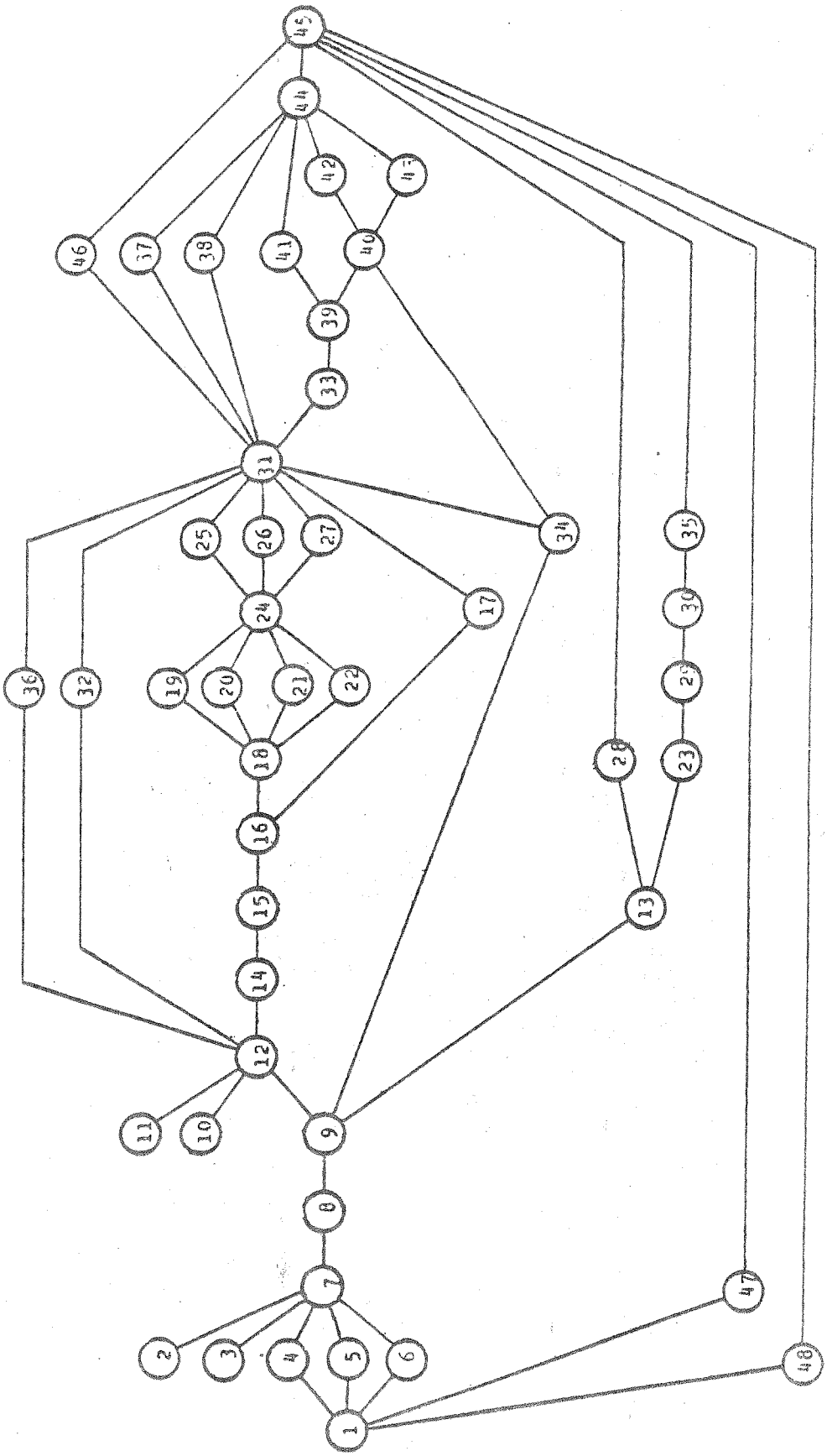


Fig. A.13. : 48 Element Problem

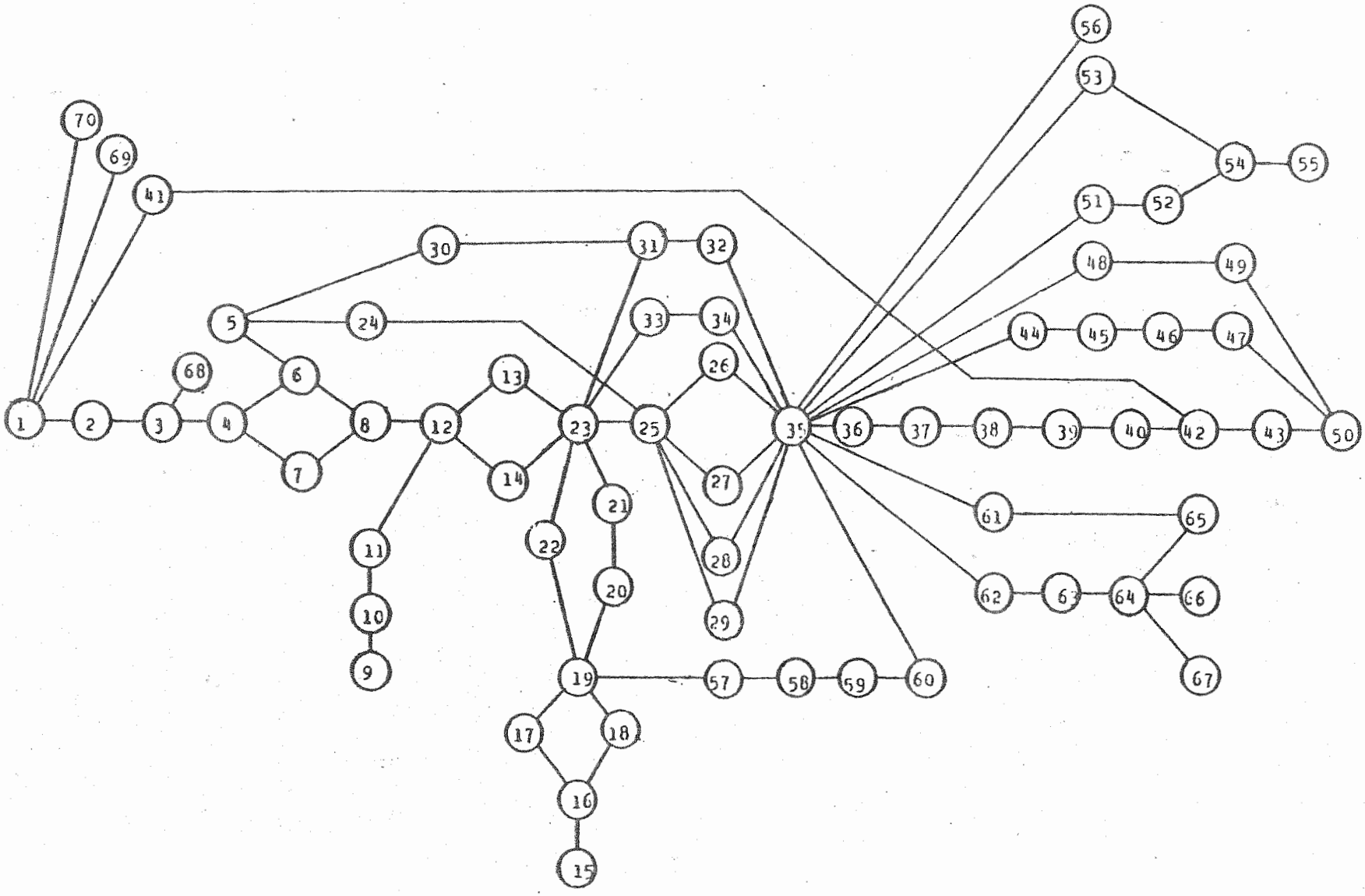


Fig. A.14 : 70 Element Problem

APPENDIX B : Generation of all feasible sequences

A very simple and rather performing algorithm is to take each of the elements without predecessors, one at the time, and restart the algorithm (recursively) for the precedence network without the given element until only 1 (the final) element remains[1]. This is equivalent to the traversal of the tree diagram of feasible sequences.

Consider, e.g. the following 7 element problem, fig. B.1. (Reeve [11], p. 84).

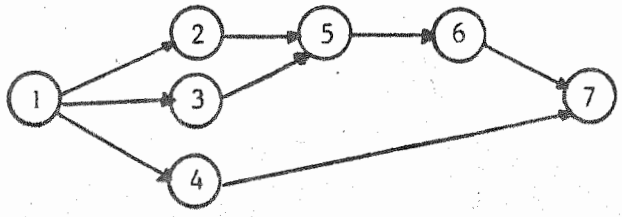


Fig. B.1.

Recursive application of the algorithm generates all 10 feasible sequences as illustrated in fig. B.2.

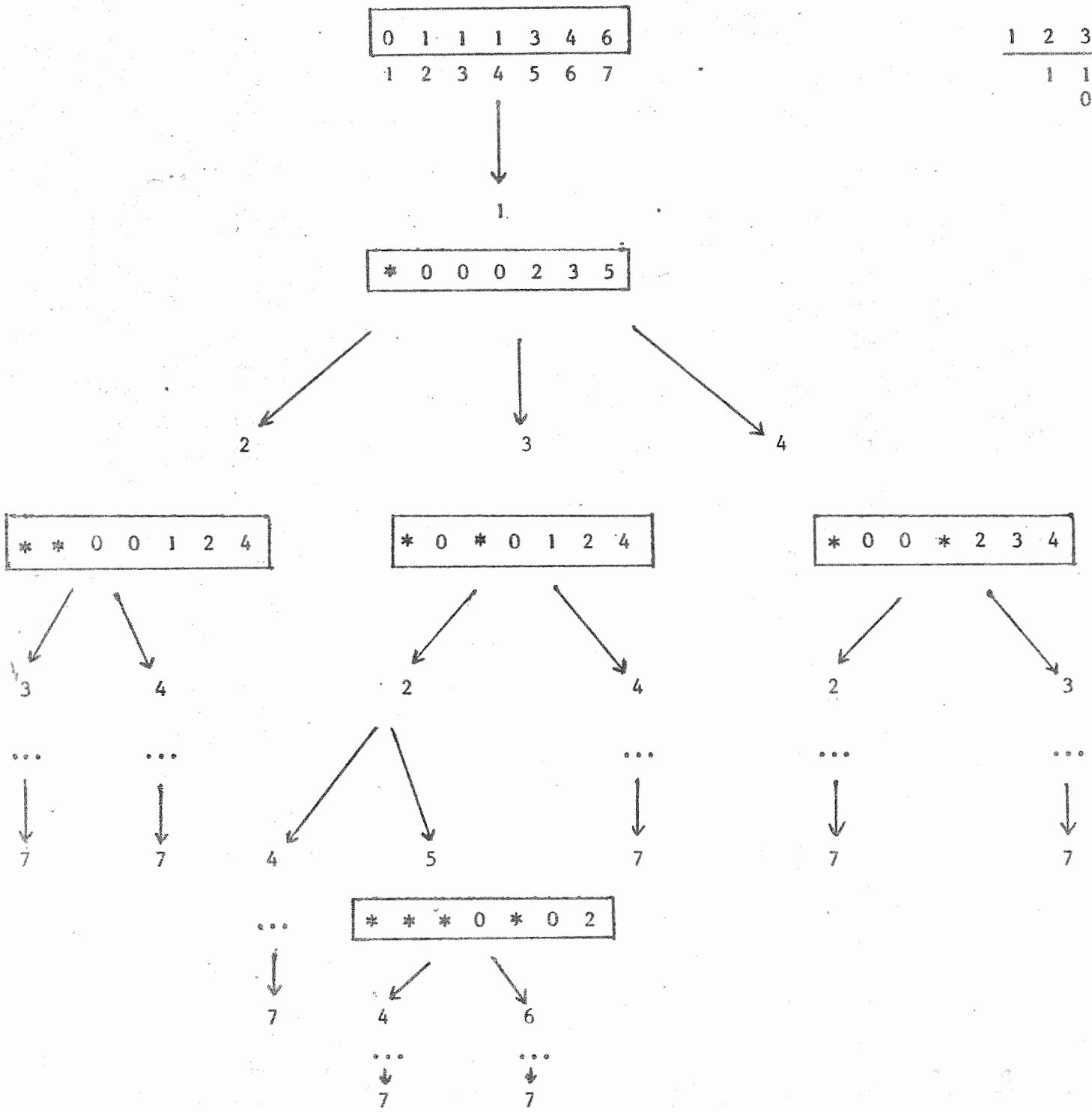
If a (sub)network has only one parent (element with no predecessors), the element could simply be deleted and the procedure restarted with the remaining number of elements. This would save the overhead of a recursive call, but slightly complicate the enumeration procedure.

Implementation of this algorithm can be done by using the precedence matrix[2]. Elements with no predecessors are then easily detected because the corresponding column contains all zeroes. Dropping an element is simply performed by deleting its row and column.

---

(1) Note that this algorithm corresponds to the general algorithm for calculating the number of feasible sequences (section 3.2). However, as all feasible sequences have to be generated extensively, the recursion has to be complete and the acceleration factors of 2.3.1. (for series-parallel networks) can not be applied.

(2) Including redundant constraints in the precedence matrix is irrelevant in this case.



1	2	3	4	5	6	7	
	1	1	1	1	1	1	1
		0	0	1	1	1	2
			0	1	1	1	3
				0	0	1	4
					1	1	5
						1	6
							7

Fig. B.4.

REFERENCES

[1] BERMAN, G., FRYER, K., Introduction to Combinatories, Academic Press, Inc., New York, 1972, 300 pp.

[2] COFFMAN, E.G., (ED), Computer and Job-Shop Scheduling Theory, John Wiley & Sons, Inc., New York, 1976, 299 pp.

[3] COMTET, L., Advanced Combinatories ,D. Reidel Publishing Company, Dordrecht, Holland, 1974, 343 pp.

[4] CONWAY, R., MAXWELL, L., MILLER, L., Theory of Scheduling, Addison-Wesley Publishing Company, Reading,Mass., 1967, 294 pp.

[5] ELMAGHRABY, S.E., HERROELEN, W.S., On the Measurement of Complexity in Activity Networks, European Journal of Operational Research, 5 (1980), pp. 223-234.

[6] ———— ,——, OR Report No. 121, Graduate Program in Operations Research, North Carolina State University, Raleigh (September 1978), 66 pp.

[7] LAWLER, E., Combinatorial Optimization, Networks and Matroids, Holt, Rinehart and Winston, New York, 1976, 374 pp.

[8] MOODIE, C.L., A Heuristic Method of Assembly Line Balancing for Assumptions of Constant or Variable Work Element Times, Ph. D. Thesis, Purdue University, 1964.

[9] NIJENHUIS, A., WILF, H., Combinatorial Algorithms, Academic Press, Inc., New York, 1975, 253 pp.

[10] PAPADIMITRIOU, C.H., STEIGLITZ, K., Combinational Optimization : Algorithms and Complexity, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982, 496 pp.

[11] REEVE, N., Balancing Continuous Stochastic Assembly Lines, Ph. D. Thesis, State University of New York, 1971.

[12] TONGE, F., Summary of a Heuristic Line Balancing Procedure, Management Science, Vol. 7, No. 1, 1960, pp. 21-42.

[13] TRAUB, J.F., Algorithms and Complexity, New Directions and Recent Results, Academic Press, Inc., New York, 1976, 523 pp.

[14] VANTHIENEN, J., An Algorithm for the Optimal Contraction of Top-Down Readable Decision Tables with Given Condition Order, Onderzoeksrapport Nr. 8210, K.U. Leuven, Department of Applied Economic Sciences, 1982, 28 pp.